探索編 α-β法

Phase4

フェーズの目標

このフェーズでは、Phase3で実装したミニマックス法を効率化させます。

実はミニマックス法は非常に非効率的な探索です。

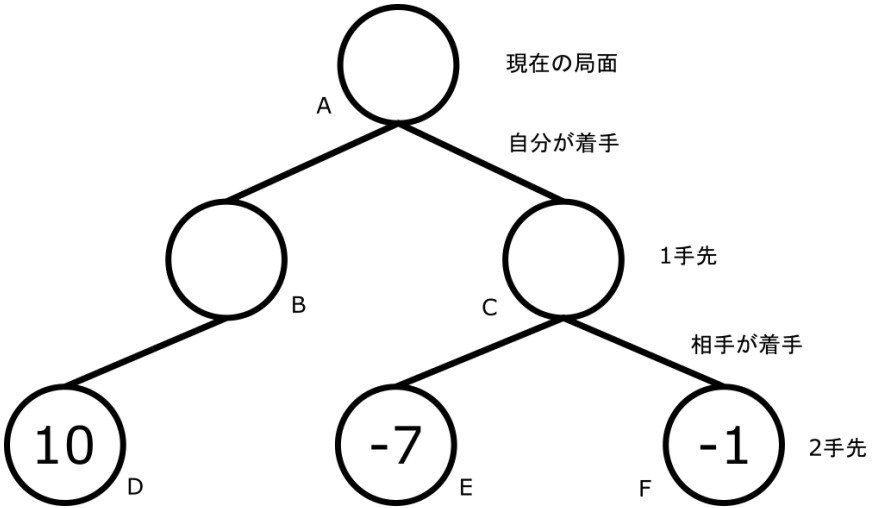
α-β法（Alpha-Beta Pruning）はその効率化手法となります。

高速化できる例

A->B->D
A->C->E

A->C->F

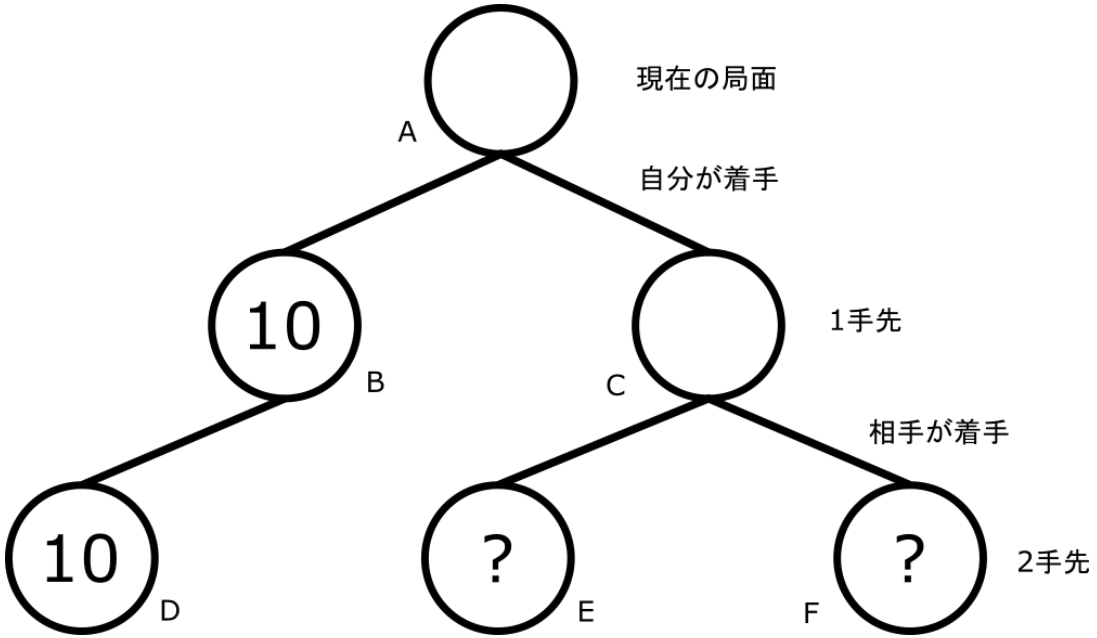
の順番で処理が進んでいきます。



まず、A->B->Dと辿ります

ノードDで評価値を得て、それを
ノードBに伝播します。

ここで、ノードCについて**「興味がある範囲」**を考えます。



ノードCはノードBの「10」を超えるかどうか

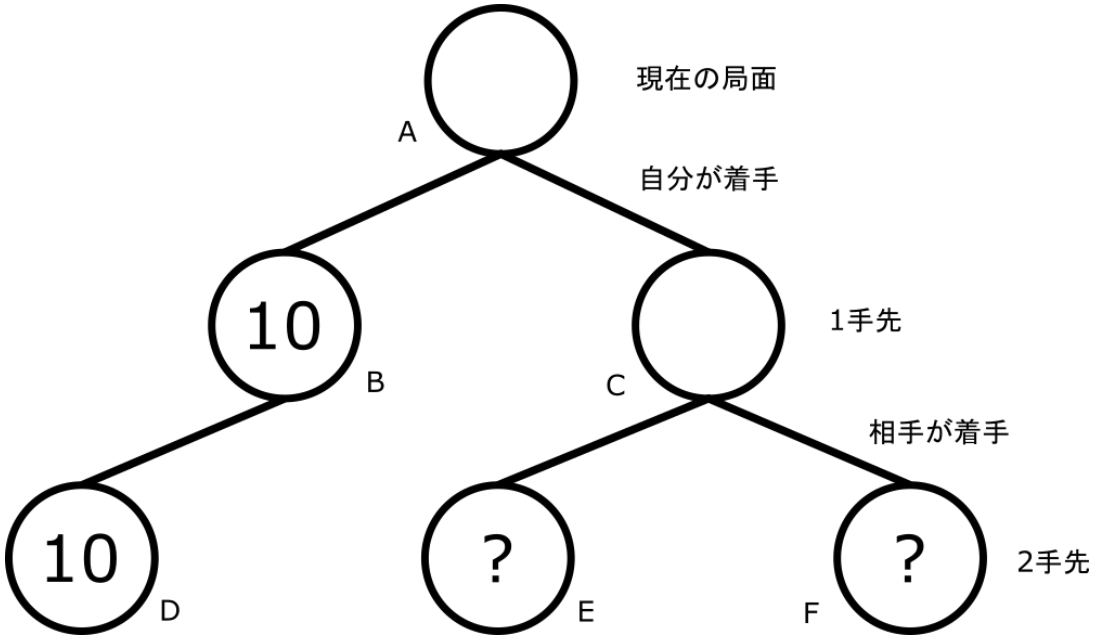
第一手目は自分の番なので

BとCの大きい方を取ります。

第二手目は相手の番なので

小さい方を取ります。

つまり、第二手目の内、1つでも10以下の評価があればノードCは10以下となります。



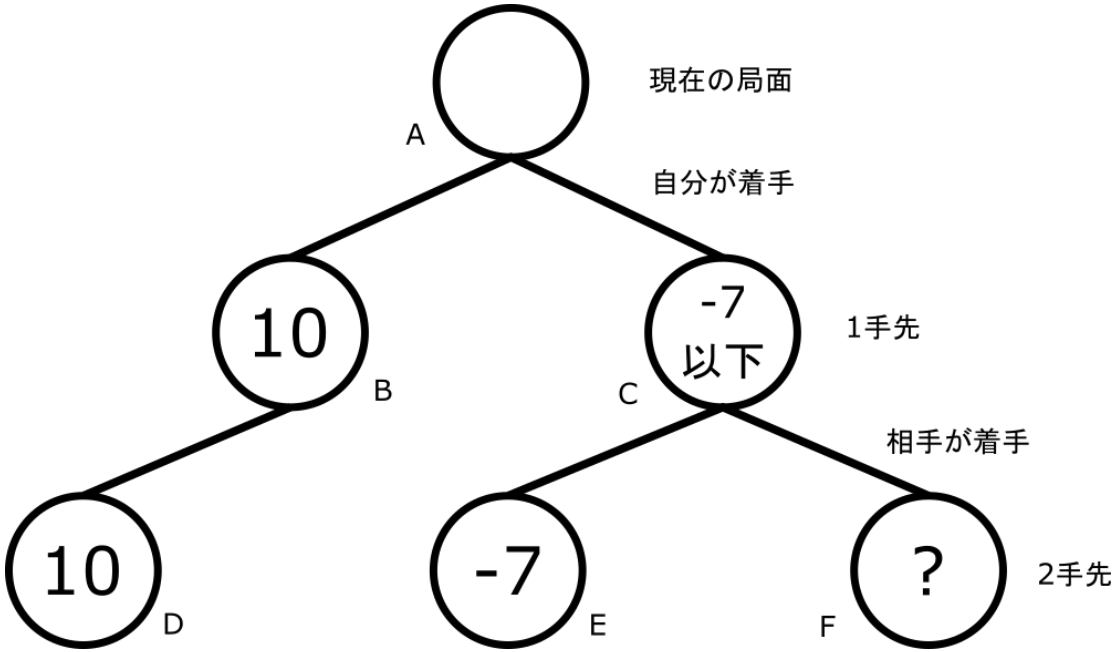
ノードEが10以下だったらノードFの評価は不要

第二手目は相手の番なので、

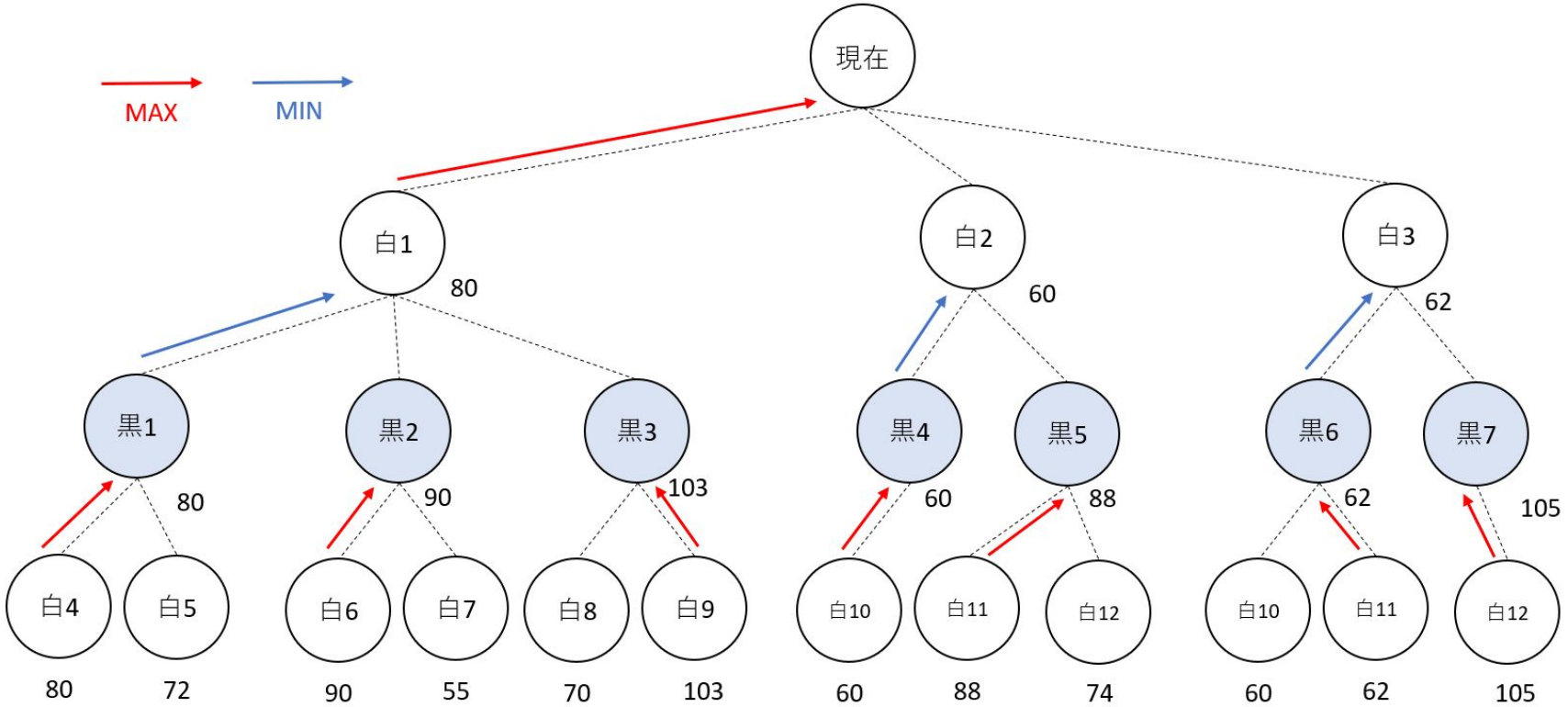
一番小さい値を上に伝播します。

ノードEが10以下であった時点でそれより大きい数字が上に伝播することはなくなります。

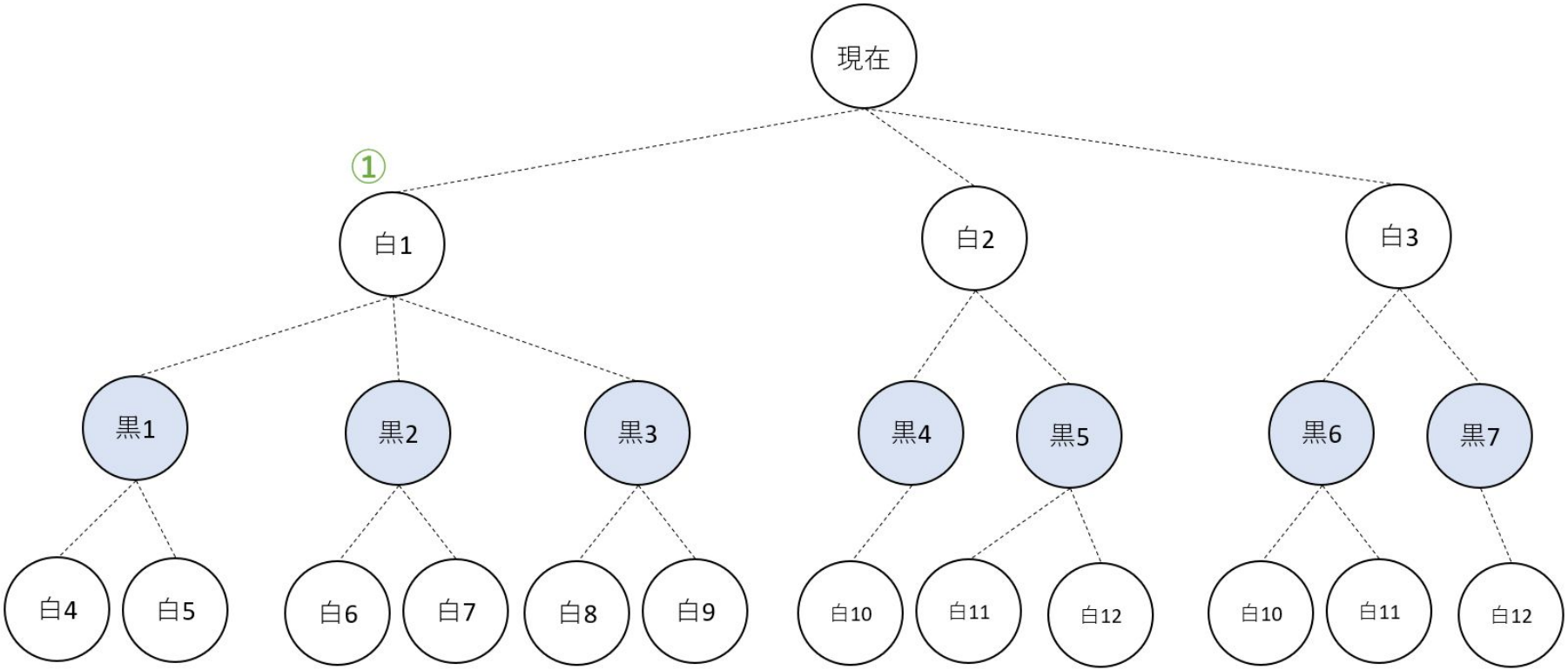
よってノードFの評価は不要となります。



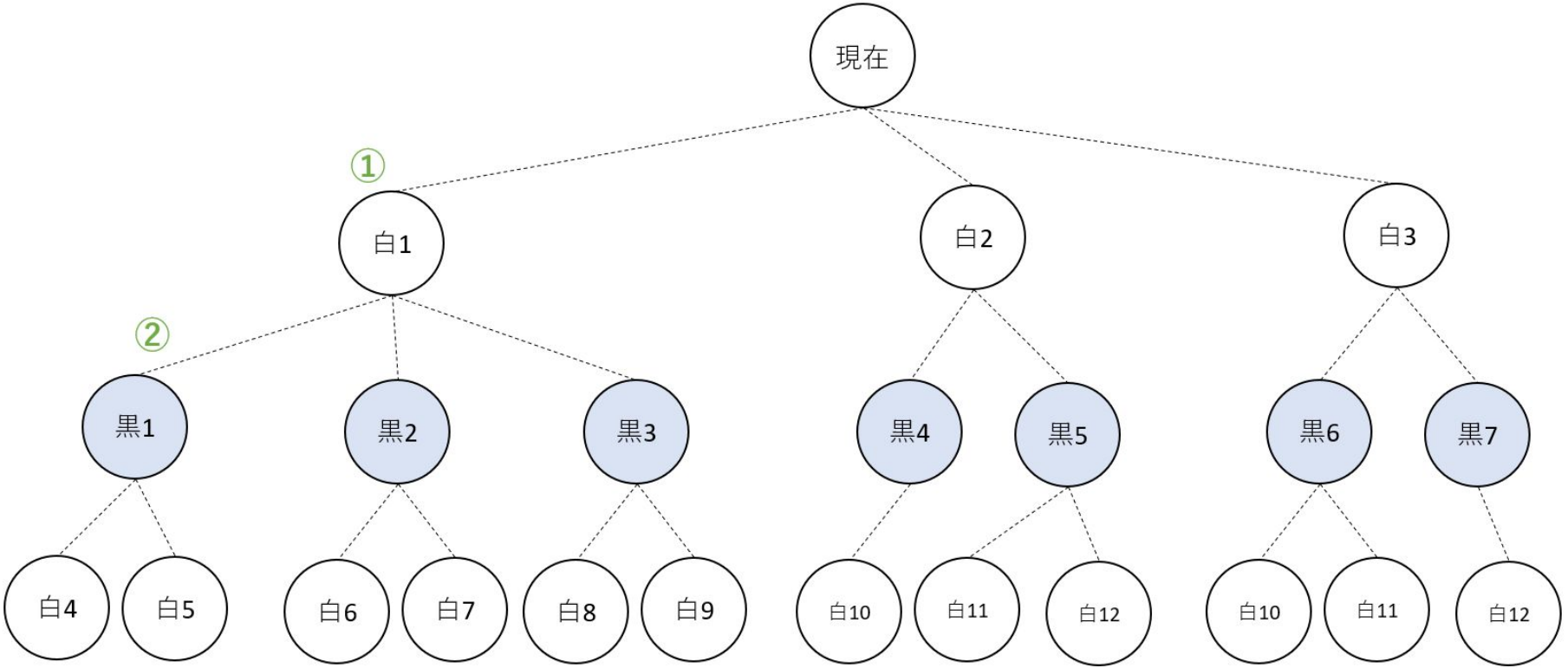
例：Phase3で見た「白番で3手先まで読む」ケース。



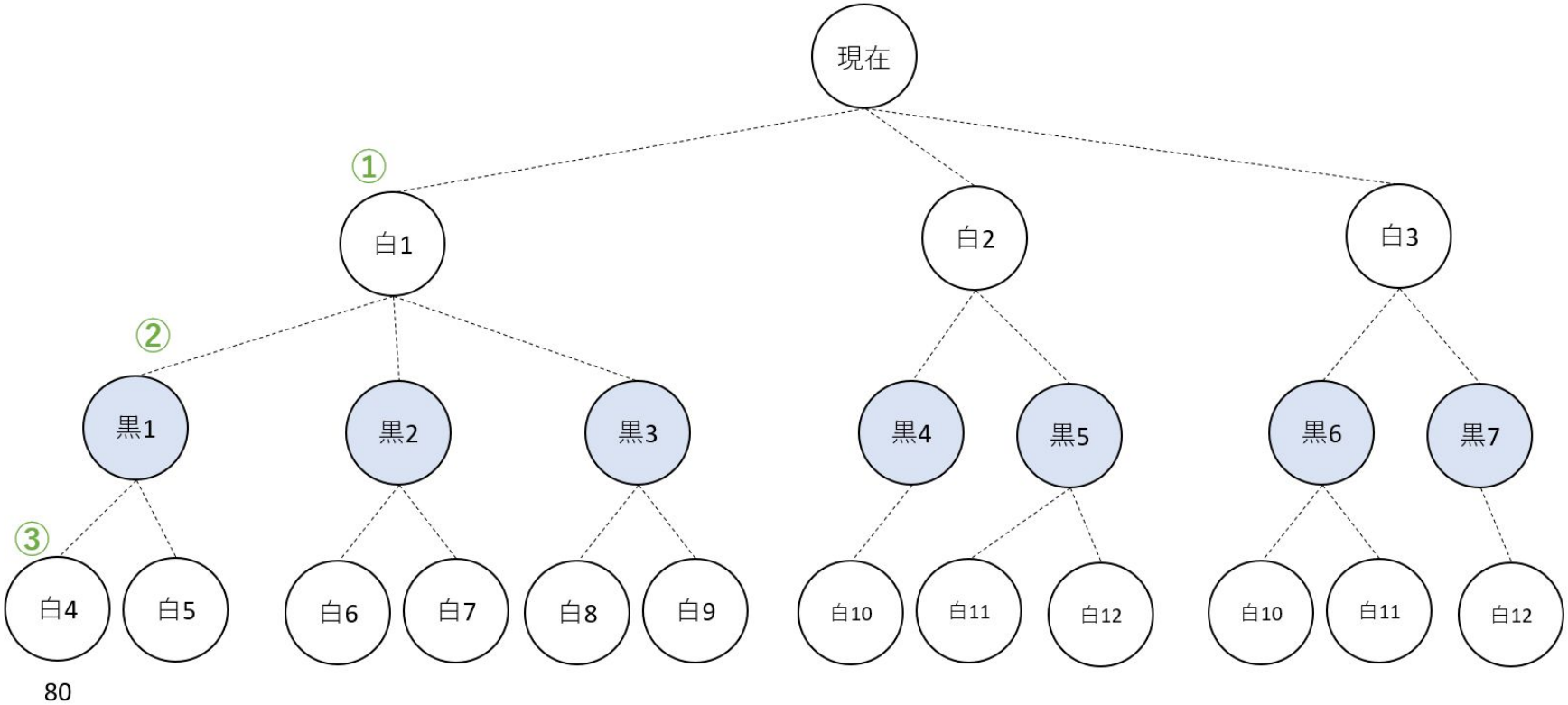
ステップ① 白1を見る


現在
①
白1
白2
白3
黒1
黒2
黒3
黒4
黒5
黒6
黒7
白4
白5
白6
白7
白8
白9
白10
白11
白12
白10
白11
白12

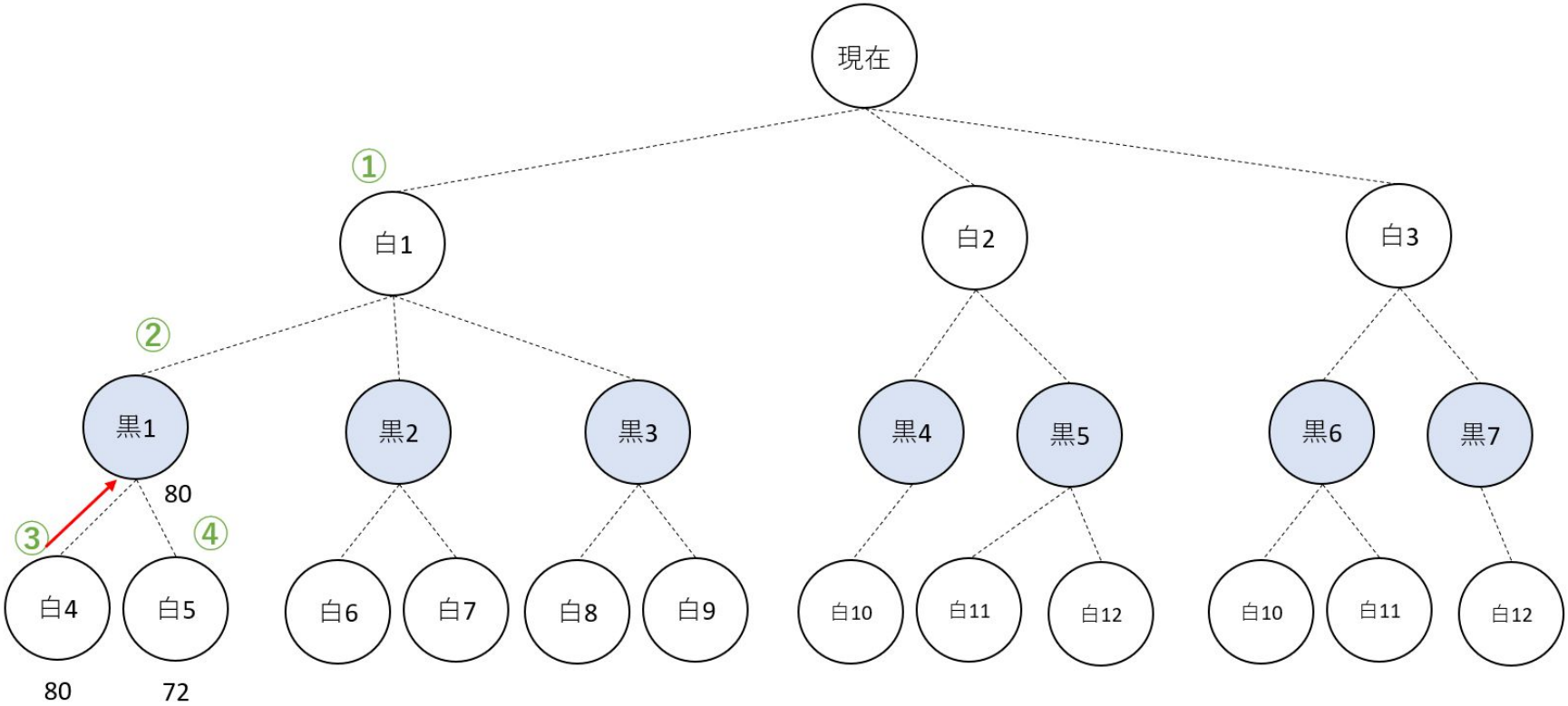
ステップ② 白1の先である黒1を見る



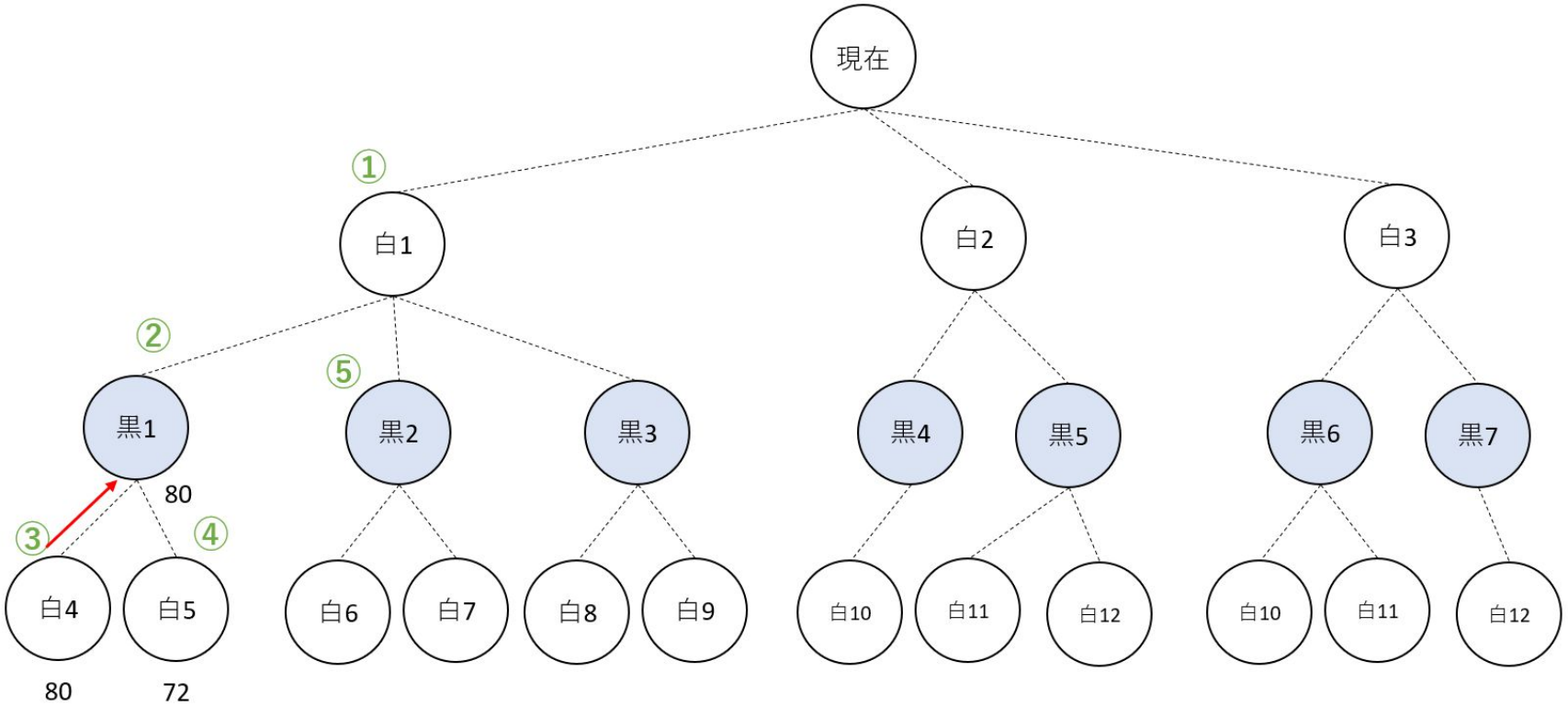
ステップ③ 黒1の先の白4を見る



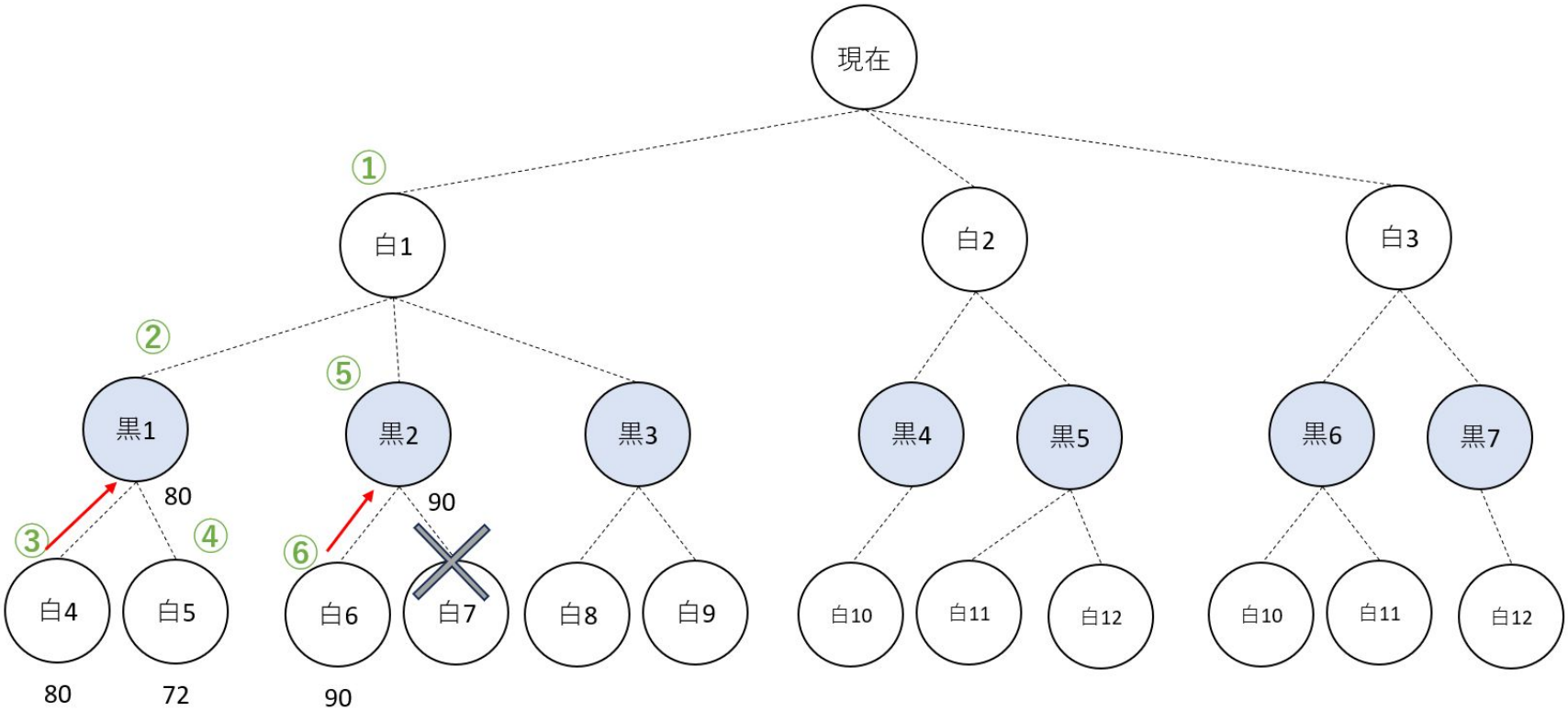
ステップ④ 黒1の先である白5を見る。白4の80を取る



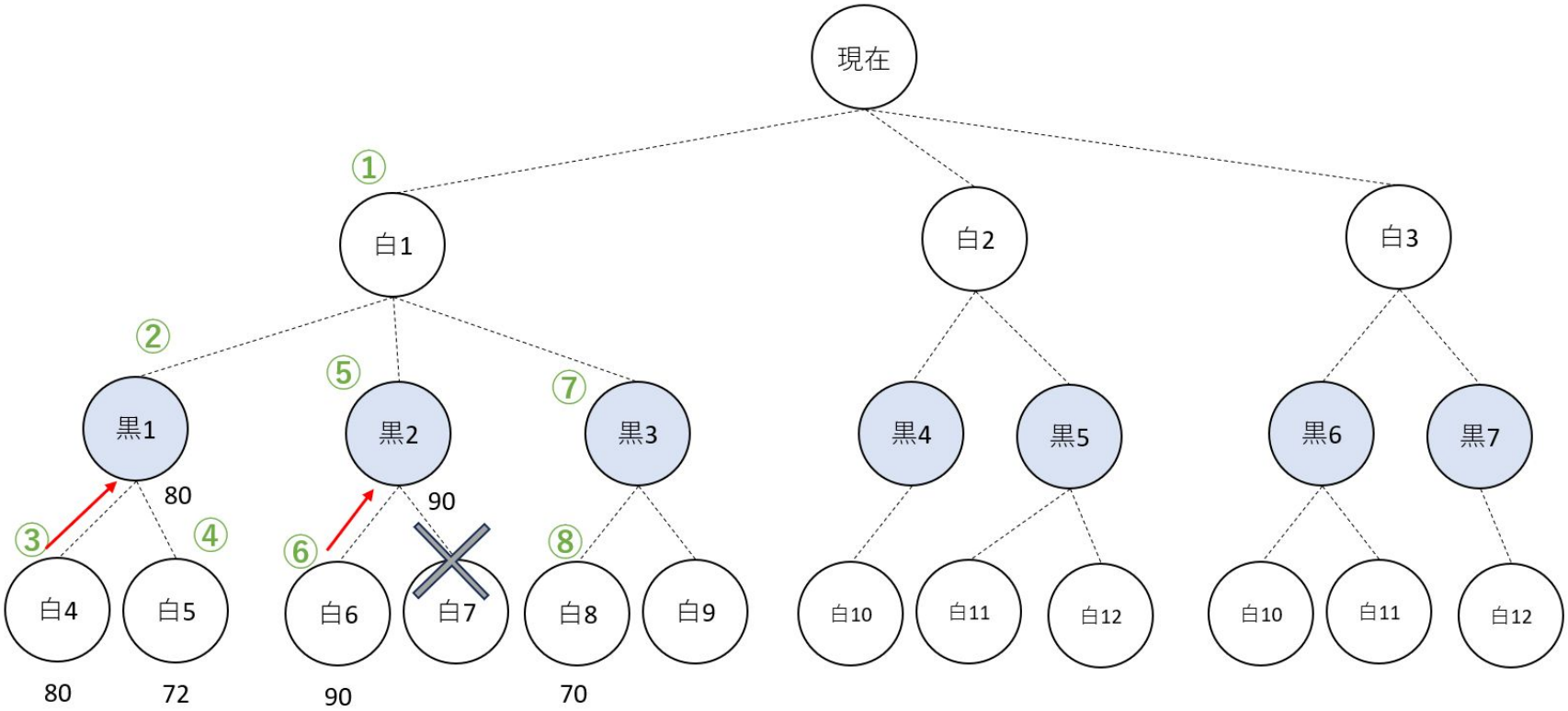
ステップ⑤ 黒2を見る。黒1の80より大きい数字は無視。



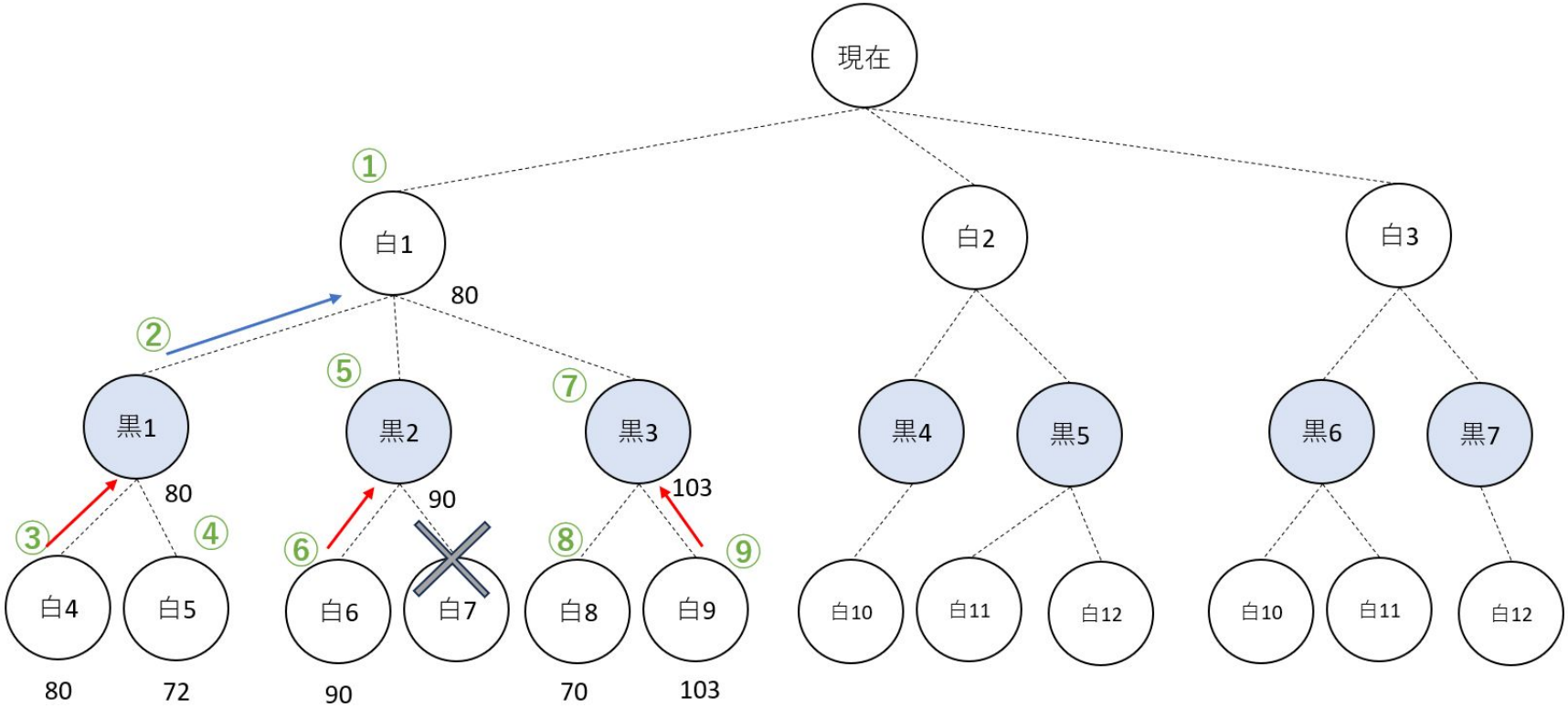
ステップ⑥ 黒2の先の白6を見る。黒1の80より大きい、白7は飛ばす



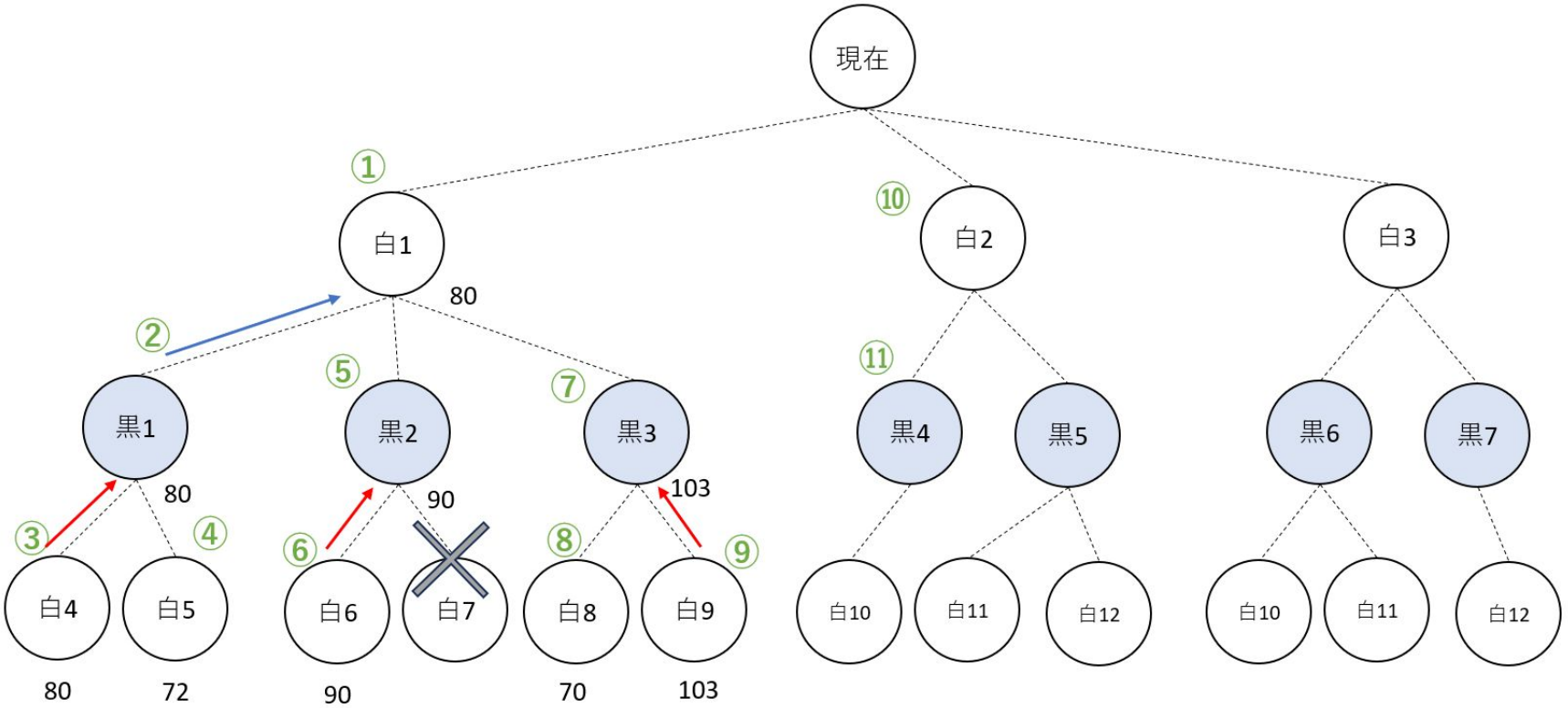
ステップ⑦⑧ 黒3の白8を見る。黒1の80より小さいので無視しない。



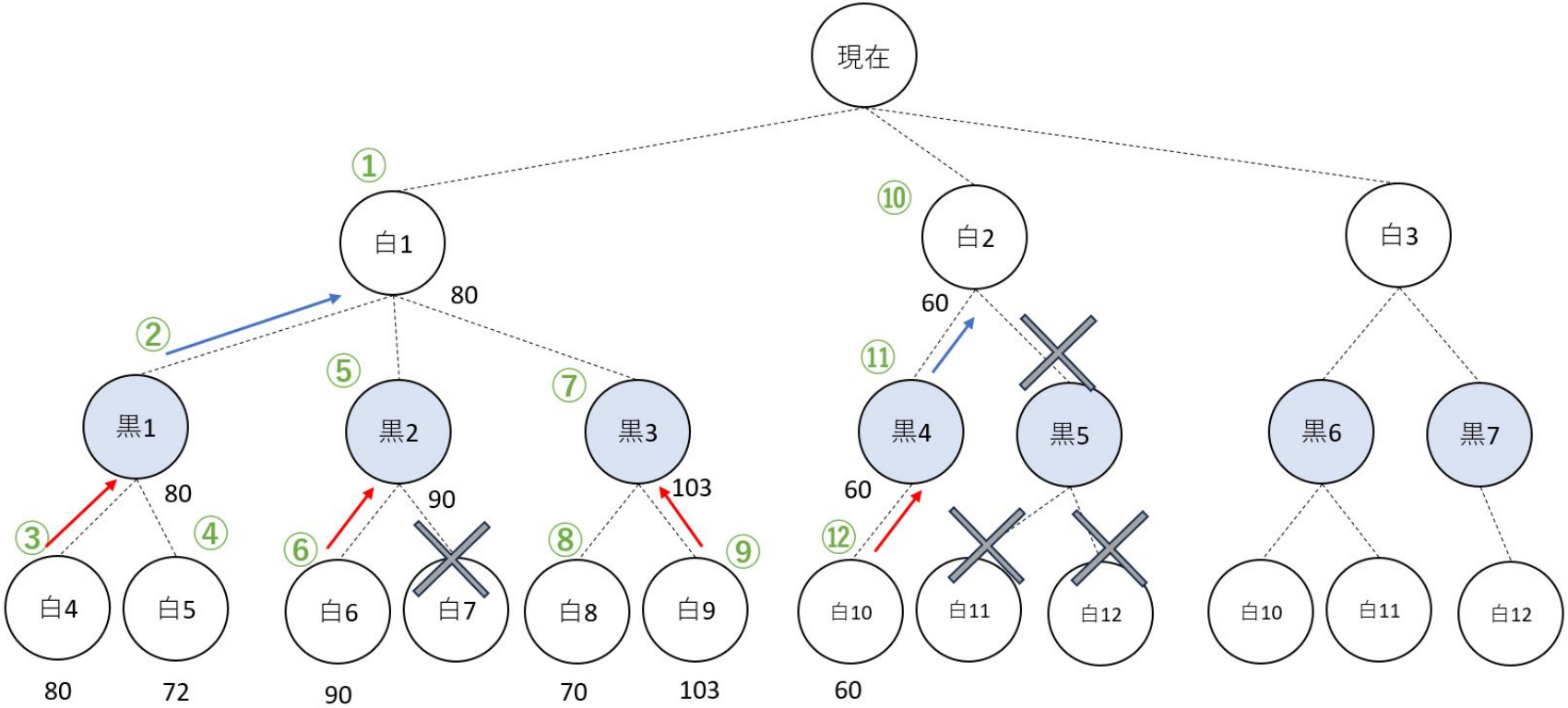
ステップ⑨ 白9を見て黒3が決まり、白1も決まった



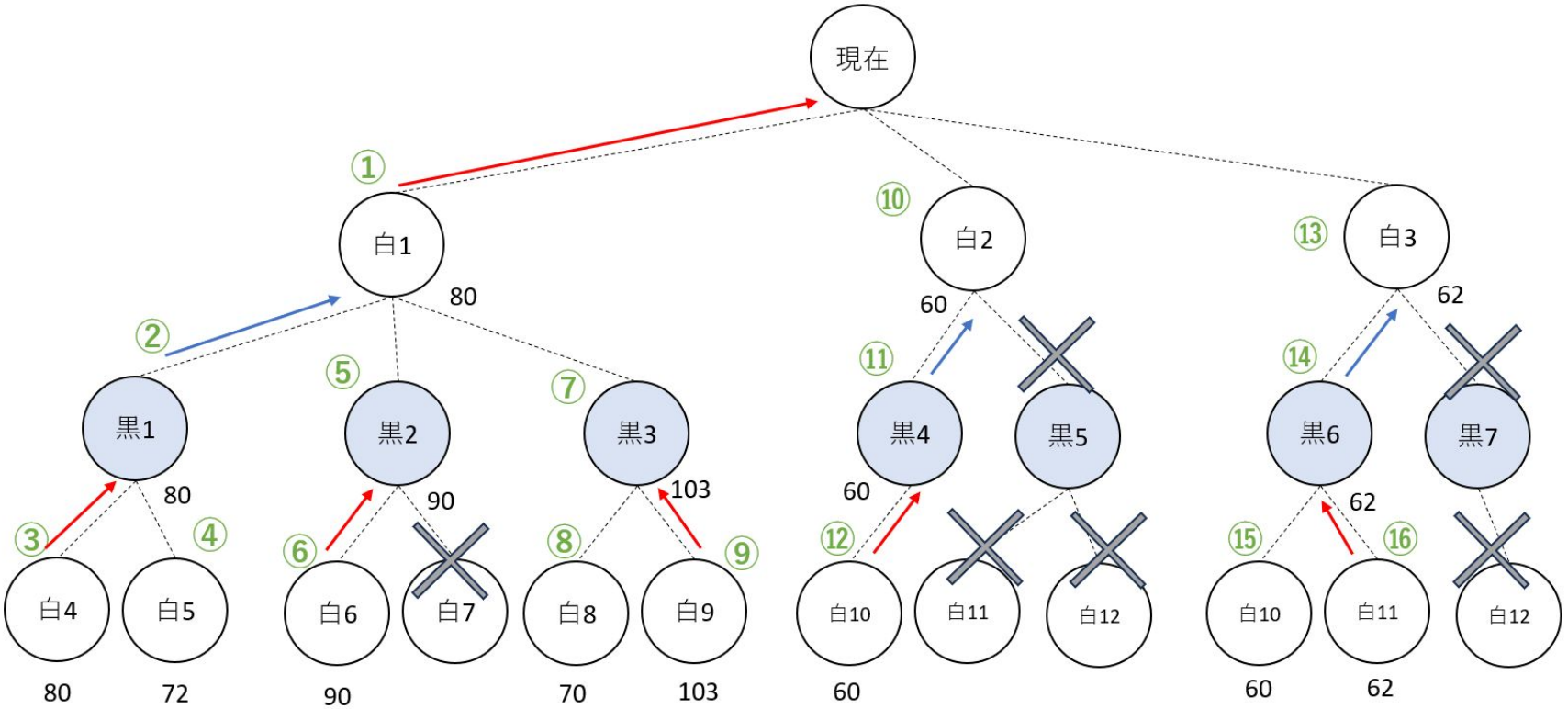
ステップ⑩⑪ 白1の80より小さい値には興味がない



ステップ⑫ 白10を見る。黒4が60なので白2は60以下で確定



ステップ⑫～⑯ 黒6が62なので白3は62以下が確定



実装のサンプル
（自信のある人はサンプルを見ずに実装してみよう)

minmax関数にα-βを実装

```
// 評価を返す再帰関数
private int minimax(Board board, int depth, int alpha, int beta, boolean isMaximizingPlayer) {
    // 終了条件：深さ0 または ゲーム終了
    if (depth == 0 || board.isTerminal()) {
        return evaluateBoard(board);
    }

    List<String> moves = board.getLegalMoves(isMaximizingPlayer ? board.getMyColor() : board.getOpponentColor());

    if (isMaximizingPlayer) {
        // Maximizer（自分）のターン：評価値の最大化を目指す
        int maxEval = Integer.MIN_VALUE;
        for (String move : moves) {
            Board childBoard = board.applyMoveAndClone(move);
            int eval = minimax(childBoard, depth - 1,  alpha, beta, false); // 次はMinimizerへ
            maxEval = Math.max(maxEval, eval);

            // α-β剪定
            alpha = Math.max(alpha, maxEval);
            if (beta <= alpha) break; // 枝刈り：betaを上回ったら探索終了

        }
        return maxEval;
    } else {
        // Minimizer（相手）のターン：評価値の最小化を目指す
        int minEval = Integer.MAX_VALUE;
        for (String move : moves) {
            // ... (Minimizer側のロジックを実装) ...

            // α-β剪定
            beta = Math.min(beta, minEval);
            if (beta <= alpha) break; // 枝刈り：alphaを下回ったら探索終了
        }
        return minEval;
    }
}
```